Fast minimum float computation in activity networks under interval uncertainty

Thierry Garaix, Christian Artigues, Cyril Briand

Fast minimum float computation in activity networks under interval uncertainty

Thierry Garaix[1,2,3], Christian Artigues[1,2] and Cyril Briand[1,2]

[1] CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM ; LAAS ; F-31077 Toulouse, France
[3] D.A.I., Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Torino, Italy

Abstract

This paper considers the following basic problem in scheduling under uncertainty: given an activity-on-node network where each activity has an uncertain duration represented by an interval, compute the minimum float of each activity over all duration scenarios. For solving this NP-hard problem, Dubois *et al.* [5] and Fortin *et al.* [7] have recently proposed an algorithm based on path enumeration. In this paper, we establish structural properties of optimal solutions and a new lower bound allowing us to design an efficient branch-and-bound procedure. We also propose two mixed integer programming formulations. The methods are compared experimentally on a large variety of randomly generated problem instances. The results show that the proposed branch-and-bound procedure is very fast and consistently outperforms the MIP formulations and the path enumeration algorithm.

1 Introduction

An important part of recent scheduling research aims at tackling uncertainty, which occurs, under various forms, in practical applications. Dealing with activity scheduling, a basic form of uncertainty involves uncertain durations. There are several ways to model such uncertainty. The duration of each activity can be seen as a random parameter is given by a distribution of probability, which falls in the category of stochastic scheduling [11, 1]. However, when probability distributions are not available, another category of approaches, gathered under the robust scheduling category, considers uncertainty scenarios and aims at providing performance guarantee on a worst-case analysis basis [10, 8].

In the latter category, we consider interval uncertainty for activity duration. Each activity has a duration that belongs to a given closed interval, which yields to an infinite number of scenarios [9]. We consider in this context one of the simplest scheduling problem: the (resource-unconstrained) project scheduling problem with simple precedence constraints, (or PERT scheduling). This central problem consists in computing the activity earliest start times, latest start times and total floats. When durations are precisely known, these values are easily computed by solving longest path problems in the corresponding activity-on-node (or activity-on-arc) network. Under interval uncertainty, the underlying best- and worst-case scenario analysis aims at computing the minimum and maximum values of the activity earliest start times, latest start times and floats. Indeed, it is of interest for a decision maker to know what would be the optimistic of pessimistic latest start time of an activity without questioning the announced project completion time. Similarly, computing the minimum float of an activity allows to known its degree of potential criticality under the worst-case scenario. According to Chanas and Zielínski [2], computing the minimum float of an activity is strongly NP-hard, and remains NP-hard in an acyclic planar network of node degree 3 [3], all other problems remaining polynomial. Complexity results and their origin are synthesized in a recent paper by Fortin *et al* [7].

In the present paper, we focus on the NP-hard minimum float computation problem. In [5], Dubois *et al.* propose an algorithm based on path enumeration. Computational experiments reported in [7] show the so-called path algorithm works well for low-to-medium density precedence networks but its performance collapses for high density networks. In this paper, we exhibit structural properties of optimal solutions that allow us to propose alternative methods, able to compute efficiently the minimum float for all types of networks, including high density precedence networks.

In Section 2 we formally define the minimum activity float problem. Section 3 recalls some previously established structural properties of optimal solutions of the problem and proposes new ones. Section 4 presents the path algorithm of Dubois *et al* and the new proposed methods which are an improved variant of the path algorithm, a branch-and-bound method and two mixed-integer programming formulations. Section 5 is devoted to intensive computational experiments to compare the different algorithms. Concluding remarks are drawn in Section 6.

2 Notations and problem definition

2.1 Activity-on-node network

A directed and acyclic activity-on-node network $G(V, A)$ is considered. The set of nodes $V = \{0, \ldots, n+1\}$ represents activities and the set of arcs A represents simple precedence relations among the activities.

2.2 Duration scenarios

Let d_i denote the duration of an activity $i \in V$. We assume d_i is an uncertain parameter belonging to a given interval $[d_i^{\min}, d_i^{\max}]$. As 0 and $n+1$ correspond to dummy start and end activities, respectively, we consider that $d_0^{\min} = d_0^{\max} = d_{n+1}^{\min} = d_{n+1}^{\max} = 0$. Let $\mathcal{D}$ denote the set of possible realizations of duration vector d. We have

$$\mathcal{D} = \{d \in \mathbb{R}^{n+2} | d^{\min} \leq d \leq d^{\max}\}.$$

2.3 Paths

A path in G is an ordered set of activities $(i_1, i_2, \ldots, i_z)$ of V^z, $z \geq 1$ where $(i_q, i_{q+1}) \in A$ for $z \geq 2$ and $1 \leq q \leq z-1$. For ease of notation, a path made of a single activity $i \in V$ may be denoted i in place of (i).

Let $\mathcal{P}$ denote the set of paths in G and let $\mathcal{P}_i$ denote the set of paths p in $\mathcal{P}$ such that $i \in p$. We also define $\mathcal{P}_i^+$ as the set of paths starting with i and $\mathcal{P}_i^-$ as the set of paths terminating at i. $\mathcal{P}_{i,j}$ is the set of paths from i to j, i.e. $\mathcal{P}_{i,j} = \mathcal{P}_i^+ \cap \mathcal{P}_j^-$. Let $o(p)$ ($e(p)$) denote the first (last) activity of path p, respectively. For ease of notation a path p may also be denoted $(s_1, s_2, \ldots, s_r)$, where each s_q is a path and $e(s_q) = o(s_{q+1})$ for $q = 1, \ldots, r-1$. For example, consider paths $p = (1, 3, 4)$ and $s = (3, 4)$, $(1, s)$ is a possible notation for $(1, 3, 4)$.

2.4 (Longest) path lengths, start times and floats

Consider a realization (or scenario) $d \in \mathcal{D}$. We define the following values as functions of scenario d. The length of a path p is equal to the sum of the duration of the activities located on p, except the last-one i.e.:

$$l_p(d) = \sum_{i \in p \setminus e(p)} d_i$$

The length of the longest path from activity a to activity b for scenario d is denoted by

$$l_{a,b}^*(d) = \max_{p \in \mathcal{P}_a^+ \cap \mathcal{P}_b^-} l_p(d)$$

The earliest start time of an activity, denoted est_i is equal to the length of the longest path terminating at i.

$$est_i(d) = l_{0,i}^*(d) = \max_{p \in \mathcal{P}_i^-} l_p(d)$$

The latest start time of an activity, denoted lst_i is equal to the length of the longest path in G (the minimum project duration) minus the length

of the longest path from i to $n+1$ (minimum elapsed time from the start of i to the end of the project).

$$lst_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d) = \max_{p \in \mathcal{P}} l_p(d) - \max_{p \in \mathcal{P}_i^+} l_p(d)$$

The float of an activity i for realized duration vector d is equal to the difference between the lastest start time and the earliest start time of the activity.

$$f_i(d) = lst_i(d) - est_i(d)$$

Equivalently, the float can be expressed as the difference between the length of the longest path in G and the length of the longest path in G traversing i.

$$f_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d) - l^*_{0,i}(d) = \max_{p \in \mathcal{P}} l_p(d) - \max_{p \in \mathcal{P}_i} l_p(d)$$

Given a scenario d, all the above-defined values can be computed in polynomial time by computing the relevant longest paths.

2.5 Problem statement

We consider the n following minimization problems that amount to compute the minimum float of each activity over the scenario set:

$$\min_{d \in \mathcal{D}} f_i(d), \quad \forall i \in V \setminus \{0, n+1\}.$$

As already mentionned the problem is strongly NP-hard [2, 3].

3 Structural properties of optimal solutions

3.1 Extreme scenarios

We first recall some previously established structural properties of optimal solutions. Let $\hat{\mathcal{D}}$ denotes the set of *extreme scenarios*, i.e.:

$$\hat{\mathcal{D}} = \left\{ d \in \mathcal{D} | \forall i \in V, d_i \in \{d_i^{\min}, d_i^{\max}\} \right\}$$

Dubois *et al* [6] established the following fundamental property

Proposition 1 ([6])**.** *For each $i \in V$, there always exists an optimal extreme scenario:*

$$\min_{d \in \mathcal{D}} f_i(d) = \min_{d \in \hat{\mathcal{D}}} f_i(d)$$

This is an important result since the solution space, initially containing an infinite number of scenarios, can be reduced to a set of 2^n scenarios.

3.2 Path-induced extreme scenarios

Given a path p, let $d^{\max}(p)$ denote the *path-induced extreme scenario* where all activities of $i \in p$ are set to $d_i^{\max}$ whereas all activities $i \in V \setminus p$ are set to $d_i^{\min}$.

If p' is the longest path in G while p denote the longest path in G traversing an activity i, recall the float $f_i(d)$ for a scenario d is equal to the length of p' minus the length of p. Hence, to minimize the float, one intuitively seeks to minimize the duration of the activities along p' while maximizing the duration of the activities along p. Dubois *et al* [5], show that for each candidate path p along i, there is actually a dominant path-induced extreme scenario, which is stated below in Proposition 2.

Proposition 2 ([5])**.** *For each $i \in V$, we have*

$$\min_{d \in \mathcal{D}} f_i(d) = \min_{p \in \mathcal{P}_i} f_i\left(d^{\max}(p)\right)$$

Proof. Suppose that for all $p \in \mathcal{P}_i$, $f_i\left(d^{\max}(p)\right) > \min_{d \in \mathcal{D}} f_i(d)$. Let d^* such that $f_i(d^*) = \min_{d \in \mathcal{D}} f_i(d)$ and $d^* \neq d^{\max}(p)$, $\forall p \in \mathcal{P}_i$. Let p' denote one of the longest paths in $\mathcal{P}_i$ for scenario d^*. The minimal float of i is equal to the difference between the length of the longest path in $\mathcal{P}$ and the length of p' in scenario d^*. Consider now scenario $d^{\max}(p')$, obtained by increasing to $d_j^{\max}$ all activities $j \in p'$ and decreasing to their minimum duration all other activities. p' stays one of the longest paths traversing i in scenario $d^{\max}(p')$ and its lengths strictly increases by $v = \sum_{j \in s}(d_j^{\max} - d_j^{\min})$ where s is the subset of activities in p' which were not set at their maximal duration in scenario $d^{\max}(p^*)$. All other paths in $\mathcal{P}$ cannot have their lengths increased by more by v, including the longest one. So, the float of i cannot be increased by switching to scenario $d^{\max}(p')$, which contradicts the hypothesis. □

It follows from Proposition 2 that the search for an optimal scenario can be replaced by the search for an optimal path, i.e. the path p^* that minimizes $f_i\left(d^{\max}(p)\right)$. Thus, Dubois *et al* [5] designed the so-called *path algorithm*, consisting in enumerating all paths $p \in \mathcal{P}$, and, for each of them, in computing $f_i\left(d^{\max}(p)\right)$, for all $i \in V$, through standard longest path computations. However the number of paths can be huge as the number of precedence constraints increases and, as experienced in [7], the path algorithm fails in producing solutions in reasonable time for dense networks. To illustrate this drawback, consider the network displayed in Figure 1, decomposed in ϕ levels such that each level contains q activities and each activity of a level q is a successor of each activity of level $p-1$. This graph has q^ϕ disctinct paths, so the enumeration becomes intractable as ϕ and q increase, even reasonably.

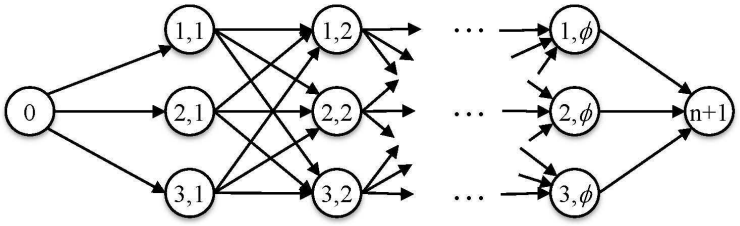

Figure 1: Intractable graph for the path algorithm

3.3 Path structure in path-induced extreme scenarios

We propose a structural analysis refining Proposition 2, so as to hopefully find further space search-reducing techniques. We first present a Corollary of Proposition 2 that allows to exclude dominated paths.

Proposition 3 (Corollary of Proposition 2)**.** *For each task i, there exists an optimal $0-n+1$ path p^* such that,*

$$l_{p^*}\left(d^{\max}(p^*)\right) = \max_{p \in \mathcal{P}_i} l_p\left(d^{\max}(p)\right).$$

Proof. This is derived from the proof of Proposition 2 since path p' constructed by the argument is the longest path traversing i under scenario $d^{\max}(p')$. □

It follows that the solution space of the minimum float problem of an activity i can be further restricted to those paths which are longest path in $\mathcal{P}_i$ for their induced extreme scenario. In the sequel, we call a *valid* path, any path verifying this property. For each valid path p, considering activity i and scenario $d\max(p)$, the float is given by

$$f_i\left(d^{\max}(p)\right) = l^*_{0,n+1}\left(d^{\max}(p)\right) - l_p\left(d^{\max}(p)\right)$$

Given a valid path p, computing the float of i requires the computation of the longest path in G and of the length of p under scenario $d^{\max}(p)$.

Actually, this longest path has a special structure, which allows simpliying the computation, as illustrated in Figure 2. Indeed, if p' denotes the longest path, p and p' are identical from activity 0 to an activity a (subpath p_a), then differ from activity a to activity b (subpaths $p_{a,b} = (p_{a,i}, p_{i,b})$ for p and $p'_{a,b}$ for p'), and coincide again from b to $n+1$ (subpath p_b). The float is then equal to the length of $p'_{a,b}$ (at minimum duration except for a) minus the length of $p_{a,b}$ (at maximum duration). This is stated formally by Proposition 4.

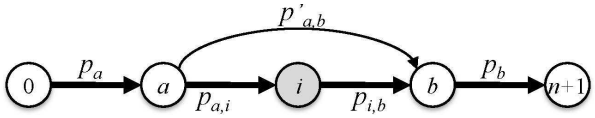

Figure 2: Structure of a valid path and of the longest path

Proposition 4. *For each activity* $\in V$ *and a valid* $0-n+1$ *path* $p = (s^-, s^+)$ *with* $e(s^-) = o(s^+) = i$*,*

$$f_i\left(d^{\max}(p)\right) = \max_{a \in s^-, b \in s^+} \{l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})\},$$

where $p_{a,b}$ *denote the subpath of* p *between* a *and* b*.*

Proof. We first show that there exists an activity pair (a, b) with $a \in s^-$ and $b \in s^+$ such that $f_i\left(d^{\max}(p)\right) = l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})$. Path p being a valid path, consider the longest path p' in $\mathcal{P}$ under scenario $d^{\max}(p)$. There are actually only two possibilities. The first one is that p and p' are identical. Then, float $f_i\left(d^{\max}(p)\right) = 0$, which is obtained by setting $a = b = i$. The second possibility is that p and p' differ. In this case, let a denote the first activity, located at or after 0 both on p and p', after which p and p' differ. Let b denote the last activity, located at or before $n+1$ on p and p', after which p and p' are identical. As illustrated on Figure 2, p and p' cannot have any activity in common from a to b and i must be located on p between a and b. Indeed, according to Bellman's principle of optimality, any other deviation from p to p' (as the deviation from u to v or the one from w to x illustrated in Figure 3) contradicts the maximality of the length of the subpath of p before or after i. Hence the slack of i under scenario $d^{\max}(p)$ is equal to

$$l_{p'_{a,b} \setminus a}\left(d^{\max}(p)\right) - l_{p_{a,b} \setminus a}\left(d^{\max}(p)\right) = l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max}).$$

We show now that for each pair (a, b) with $a \in s^-$ and $b \in s^+$, we have $f_i\left(d^{\max}(p)\right) \geq l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})$. Suppose (a^*, b^*) such that $f_i\left(d^{\max}(p)\right) < l^*_{a^*,b^*}(d^{\min}) + d^{\max}_{a^*} - d^{\min}_{a^*} - l_{p_{a^*,b^*}}(d^{\max})$. Consider the subpath s from a^* to b^* having the shortest path at minimal duration $l^*_{a^*,b^*}(d^{\min})$. The length of s under $d^{\max}(p)$ is larger than or equal to $l^*_{a^*,b^*}(d^{\min})$. So we can derive a path p'' identical to p from 0 to a^*, then following s, then identical to p from b^* to $n+1$. We have $l_{p''}\left(d^{\max}(p)\right) - l_p\left(d^{\max}(p)\right) \geq l^*_{a^*,b^*}(d^{\min}) + d^{\max}_{a^*} - d^{\min}_{a^*} - l_{p_{a^*,b^*}}(d^{\max})$. Consider now the longest $0-n+1$ path p' under $(d^{\max}(p))$. We have $f_i\left(d^{\max}(p)\right) = l_{p'}\left(d^{\max}(p)\right) - l_p\left(d^{\max}(p)\right)$. Combined with the previous expression, the length of p'' is larger than that of p', a contradiction. □

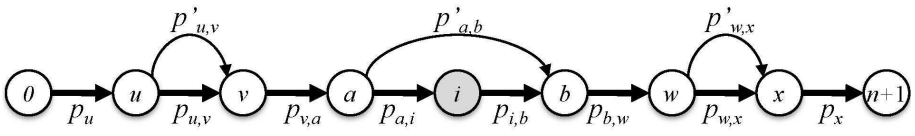

Figure 3: A dominated path structure

Finally, the problem of computing the minimal float of activity i can be stated as the search for a valid $0-n+1$ path (s^-, s^+) with $e(s^-) = o(s^+) = i$, such that

$$\max_{a \in s^-, b \in s^+} \{l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})\}$$

is minimal.

3.4 Float lower bound given a valid partial path

A direct consequence of Proposition 4 is to obtain a lower bound of the float obtained by extending a valid partial path $p_{x,y} = (s^-, s^+)$ with $e(s^-) = o(s^+) = i$, $o(s^-) = x$, $e(s^+) = y$ (see Figure 4 with $s^- = (p_{x,a}, p_{a,i})$ and $s^+ = (p_{i,b}, p_{b,y})$). We assume $p_{x,y}$ is valid in the sense that s^- is the shortest path in $d^{\max}(p_{x,y})$ between x and i and s^+ is the shortest path in $d^{\max}(p_{x,y})$ between i and y. Then, according to Proposition 4, if there exists a valid path $p = (\sigma^-, s^-, s^+, \sigma^+)$ with $o(\sigma^-) = 0$ and $e(\sigma^+) = n + 1$, it verifies

$$f_i\,(d^{\max}(p)) \geq \max_{a \in s^-, b \in s^+} \{l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})\},$$

where $p_{a,b}$ is the subpath of $p_{x,y}$ from a to b.

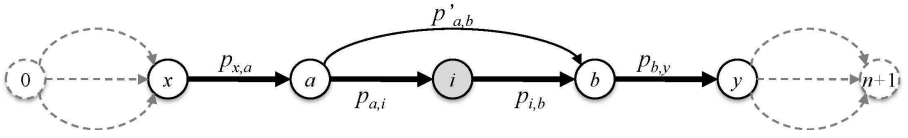

Figure 4: Partial path structure for float computation

4 Algorithms

4.1 Path Algorithm and improved variant

In this section we describe the Path algorithm proposed in [5] and an improved variant.

The proposed algorithm by Dubois *et al.* [5] computes all $0 - n + 1$ paths p, and, for each scenario $d^{\max}(p)$, the associated floats of all activities through a classical PERT-CPM algorithm.

From Proposition 2, we remark that only activities on p need to be updated since durations of other activities are set to $d^{\min}$. Now, the floats of activities set to $d^{\min}$ are necessarily not larger in a scenario that fixes their duration to $d^{\max}$ (keeping other unchanged) and, such scenarios will be necessarily evaluated since all paths are enumerated. In order to compute the float associated to activities of p, we compute the longest path p' in G under scenario $d^{\max}(p)$. The minimum floats of activities on p are updated with the upper bound $l'_p(d^{\max}(p)) - l_p(d^{\max}(p))$.

For both algorithms a topological sort is run as a preprocessing. Then, a recursive method explores each extreme scenario by building all the paths of the activity-on-node network. The pseudo-codes are detailed in algorithm 1 and 2, respectively. In the original version of the algorithm, at each complete path joining the last activity, we can expect to run $2m$ operations computing all earliest and latest starting times plus n operations updating all the floats. Our variant only runs m operations to compute the longest path (since the network is a DAG) and $|p|$ operations to update the floats of activities along p. Moreover, the longest path procedure is truncated since some partial longest paths can be reused exploring all the extreme scenarios.

Algorithm 1 original_path(p)

```
1: i := last activity of p;
2: if i = N + 1 then
3:    compute all earliest starting time est in d^max(p);
4:    compute all latest starting time lst in d^max(p);
5:    for all j ∈ V do
6:       f_j := min{f_j, lst_j − est_j};
7:    end for
8: else
9:    for all j ∈ Γ(i) do
10:      p' := (p, j);
11:      original_path(p');
12:   end for
13: end if
```

4.2 Specific branch-and-bound method

Propositions 3 and 4 allow to design an efficient branch-and-bound procedure for the computation of the minimum floats. In the procedure, the nodes of the search tree correspond to valid partial paths, related to a given activity i, and are stored inside a stack $\mathcal{Q}$ for depth-first search. Given a valid partial path $p_{x,y} = (s^-, s^+)$, with $e(s^-) = o(s^+) = i$, $o(s^-) = x$, $e(s^+) = y$, the branching scheme consists in alternatively extending the path to the left or

Algorithm 2 improved_path(p)

```
1: i := last activity of p;
2: if i = N + 1 then
3:    p' := longest path in d^max(p);
4:    for all j ∈ p do
5:       f_j := min{f_j, l'_p(d^max(p)) − l_p(d^max(p))};
6:    end for
7: else
8:    for all j ∈ Γ(i) do
9:       p' := (p, j);
10:      improved_path(p');
11:   end for
12: end if
```

to the right, by considering either all the immediate predecessors of x, or all the immediate successors of y, discarding the non-valid path extensions (with respect to Proposition 3). Thus, considering a given path-extension direction δ, a node has always as many children as immediate valid path-extensions. Each node of the search tree, memorizes also the next direction δ ($\delta \in \{\text{left, right}\}$) to consider for the path extension. A leaf of the search tree corresponds to a valid path $p \in \mathcal{P}_i$ from 0 to $n+1$ and is evaluated by $f_i\,(d^{\max}(p))$. Classically, a partial path $p_{x,y} = (s^-, s^+)$ is deleted only if its current evaluation, $\max_{a \in s^-, b \in s^+}\{l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})\}$ (see Proposition 4), is lower than the best float f_i already found.

Let us comment in details Algorithm 3. We remark first that the longest path values $l^*_{i,j}(d^{\min})$ can be precomputed very fastly using the variant of Bellman-Ford's algorithm for directed acyclic graphs. Computing of the minimum float of all activities requires to run the branch-and-bound procedure n times (see step 1). At the beginning of each run (steps 2-3), f_i is set to ∞, stack $\mathcal{Q}$ contains only a single partial path $p_{x,y} = i$, and the path-extension direction is set to left by default. The branch-and-bound procedure is implemented in steps 4-34. While stack $\mathcal{Q}$ is not empty, a partial path $p_{x,y}$ is taken from the stack, with its path-extension direction δ (step 5). Preliminarily, with respect to Proposition 4, the activities (a, b) for which the longest path from x to y differs from $p_{x,y}$ are determined (see step 6). Note that this can be done incrementally in linear time: if x (y) is the last activity to which the path has been extended to the left (right), then for a possible update of the pair (a, b) found for the father path, we only need to consider pairs (u, v) such that $u = x$ ($v = y$) and $v \in s^+$ ($u \in s^-$), respectively.

If the evaluation of $p_{x,y}$ fails (*i.e.*, $l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max}) \geq f_i$), the path is deleted (see step 7). Otherwise, if $p_{x,y}$ is a complete path (*i.e.*, $x = 0$ and $y = n+1$), then the new f_i value is memorized (see steps 8-9). If $p_{x,y}$ is not a complete path, it is extended with respect to direction

δ. We only comment here the case where δ=left (see steps 10-20), the case where δ =right (see steps 21-32) being symmetrical. If δ=left, all the immediate predecessors x' of x are considered for possible path extension (see step 11). If the partial path $p_{x',y} = (s'^{-}, s^{+})$ with $s'^{-} = (x', s^{-})$ is valid (*i.e.*, if Proposition 3 is verified, a new child is generated and pushed in $\mathcal{Q}$ (see steps 12-18). To verify Proposition 3, the obtained path $s'^{-} = (x', s^{-})$ from x' to i must remain a longest path in $d^{\max}(s'^{-})$. This is not the case if we can find an activity u such that $s^{-} = (s^{-}_{x,u}, s^{+}_{u,i})$ between x and i such that the path $(x, s^{-}_{x,u})$ has a smaller length than the deviation from x' to u via the longest path at minimal duration, i.e.

$$d^{\max}_{x'} + l_{s^{-}_{x,u}}(d^{\max}) < l^{*}_{x',u}(d^{\min}) + d^{\max}_{x'} - d^{\min}_{x'}.$$

We underline again that the validity of a path extension can be checked in linear time since all longest path at minimum duration have been precomputed. Once a new left-path-extension is made, the next extension direction is set to right unless $y = n + 1$ (see steps 12-17).

4.3 Mixed integer programming

In this section, we propose mixed integer programming (MIP) formulations exploiting also Proposition 2 and the concept of path-induced extreme scenario. For each activity, a binary variable x_i indicates if i is located on the optimal path ($x_i = 1$) or not ($x_i = 0$). The formulations also involve earliest start time continuous variables S_i, whose value is the length of the longest path from 0 to i in the scenario that minimizes the float. We propose two different models: an activity independent formulation and an activity dependent formulation.

4.3.1 Activity-independent MIP

The activity-independent formulation is solved iteratively. At iteration k, a set U_k of activities with unknown floats is considered (at iteration 0, $U_0 = V \setminus \{0, n+1\}$). MIP Q($k$) finds the minimal float for a set $F_k \subset U_k$ of activities, with its related path-induced extreme scenario $d^{\max}(p)$, where F_k is the set of activities belonging to the computed path p, (*i.e.*, such that $x_i = 1$). At the next iteration, U_{k+1} is set to $U_k \setminus F_k$. Due do the minimization objective, note that, if $i \in U_k \setminus F_k$ and if $j \in F_0 \cap F_1 \cap \cdots \cap F_k$, then $\min_{d\in\mathcal{D}} f_i(d) \geq \min_{d\in\mathcal{D}} f_j(d)$. It follows that at iteration 0, a path of activities having a null float will be issued. Then, the floats are issued in non decreasing order along the iterations, until all activities get their minimal float computed. This mechanism ensures that at most n iterations will be needed.

Q(k) :

$$\min z(k) = S_{n+1} - \sum_i d^{max}_i x_i \tag{1}$$

Algorithm 3 branch-and-bound

for all $i \in V \setminus \{0, n+1\}$ **do**
  $f_i \leftarrow \infty$;
  push$(\mathcal{P}, (i, \text{left}))$;
  while $\mathcal{P} \neq \emptyset$ **do**
    $(p_{x,y} = (s^-, s^+), \delta) \leftarrow$pop$(\mathcal{P})$;
    $(a, b) \leftarrow \underset{\{u \in s^-, v \in s^+\}}{\text{argmax}} \left(l^*_{u,v}(d^{\min}) + d^{\max}_u - d^{\min}_u - l_{p_{u,v}}(d^{\max})\right)$;
    if $l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max}) < f_i$ **then**
      if $x = 0$ AND $y = n + 1$ **then**
        $f_i \leftarrow l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a,b}}(d^{\max})$;
      else if $\delta =$left **then**
        for all $x' \in \Gamma^{-1}(x)$ **do**
          if $d^{\max}_{x'} + l_{s^-_{x,u}}(d^{\max}) \geq l^*_{x',u}(d^{\min}) + d^{\max}_{x'} - d^{\min}_{x'}$, $\forall u \in s^-$ **then**
            if $y \neq n + 1$ **then**
              $\delta \leftarrow$right;
            else
              $\delta \leftarrow$left;
            end if
            push$(\mathcal{P}, ((x', p_{x,y}), \delta)$;
          end if
        end for
      else if $\delta =$right **then**
        for all $y' \in \Gamma(y)$ **do**
          if $l_{s^+_{v,y}}(d^{\max}) + d^{\max}_y \geq l^*_{v,y'}(d^{\min}) + d^{\max}_v - d^{\min}_v$, $\forall v \in s^+$ **then**
            if $x \neq 0$ **then**
              $\delta \leftarrow$left;
            else
              $\delta \leftarrow$right;
            end if
            push$(\mathcal{P}, ((p_{x,y}, y'), \delta)$;
          end if
        end for
      end if
    end if
  end while
end for

subject to

$$z(k) \geq z(k-1), \tag{2}$$

$$S_j \geq S_i + d_i^{min} + (d_i^{max} - d_i^{min})x_i \quad \forall (i,j) \in A, \tag{3}$$

$$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{4}$$

$$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{5}$$

$$x_0 = x_{n+1} = 1, \tag{6}$$

$$\sum_{i \in U_k} x_i \geq 1, \tag{7}$$

$$x_i \in \{0,1\} \qquad \forall i \in V. \tag{8}$$

Objective (1) gives the difference between the longest path in G, given by S_{n+1}, and the length of path p represented by x_i values, on which all activities are set to their maximal duration. Constraint (2) states that the float is non decreasing between two successive iterations (this constraint is redundant). Constraints (3) implement precedence relations between activities $(i,j) \in A$, that is $S_j - S_i \geq d_i^{\max}$ if i is on p (*i.e.*, $x_i = 1$), $S_j - S_i \geq d_i^{\max}$ otherwise(*i.e.*, $x_i = 0$). Constraints (4-6) enforce the x_i values to define a path from 0 to $n+1$. Sets of predecessors (successors) of i being denoted Γ_i^{-1} (Γ_i), an activity can be located on p only if exactly one of its predecessors (successors) belongs also to p, respectively. Constraints (6) state that 0 and $n+1$ are the extremities of path p. Constraint (7) enforces the selected path to traverse at least one activity with unknown float (set U_k).

4.3.2 Activity-dependent MIP

The activity-dependent MIP P(k) below is just a particular case of Q(k), where at each iteration k, the set U_k is reduced to the single activity k. Hence, for each activity k, P(k) minimizes float f_k. To obtain all minimal floats, exactly n iterations are needed. Giving this definition, the MIP can be rewritten as follows. The only difference with Q(k) lies in constraint (14) stating that k must be located on path p.

P(k) :

$$\min f_k = S_{n+1} - \sum_i d_i^{max} x_i \tag{9}$$

subject to

$$S_j \geq S_i + d_i^{min} + (d_i^{max} - d_i^{min})x_i \quad \forall (i,j) \in A, \tag{10}$$

$$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{11}$$

$$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{12}$$

$$x_0 = x_{n+1} = 1, \tag{13}$$

$$x_k = 1, \tag{14}$$

$$x_i \in \{0,1\} \qquad \forall i \in V. \tag{15}$$

Note that computing all minimum floats with Q(k) may need less iterations than with P(k). However, the search space for P(k) is reduced since only ancestors and descendants of k may belong to p. Section 5 compares experimentally the two formulations together with the proposed specific branch-and-bound method and the path algorithm of Dubois *et al.* [5] and Fortin *et al.* [7].

5 Computational experiments

This section presents computational results to compare the efficiencies of the Path algorithm (Path) of Dubois *et al.* [5] and Fortin *et al.* [7] (original_path), its improved variant (improved_path), the activity dependent (MIPd) and independent (MIPi) mixed-integer programs and our specific branch-and-bound procedure (BB). All the experiments were performed on the same Intel 2.6 GHz processor having 3 Go RAM running on a Linux XUbuntu operating system. The MIP approaches were implemented using Cplex 12.0.

5.1 Randomly generated problem instances

Two sets of robust-PERT problem instances were used. The first set, also considered in [7], is built up from the 120 problem instances of the PSPLIB, that was originally designed for comparing solving approaches for the Resource-Constrained Project Scheduling Problems (RCPSP). Each problem instance is defined by an activity network, each activity i being characterized by a deterministic duration d_i and a resource consumption vector. As in [7], for building up robust-PERT instances, we omit the resource characteristics and the interval uncertainty of the activity duration were set to $[d_i, d_i \times 1.2]$. These benchmarks are named $PSPLIB-$instances.

As in [7], using RanGen1 software ([4]), we also generated a set of 9 series of 100 activities networks called *RG*-instances, each of them being identified by its order strength. The order strength measures the density

of directed acyclic graphs. Therefore, networks with a high order strength have a high number of paths. An order strength of 0 means a full parallelism while an order strength of 1 gives a total order. We parametrized RanGen1 for order strengths ranging from 0.1 to 0.9 by steps of 0.1. For determining the uncertainty interval $[d_i^{\min}, d_i^{\max}]$ of each activity, the benchmarking procedure described in [7] was used to generate the *RG*-instances. It suggests to randomly set a duration $d_i^{\min} \in [10, 50]$, then to set $d_1^{\max}$ to $d_i^{\min} \times 1.2$ (as done with the $PSPLIB-$instances).

Nevertheless, we point out that this interval generation scheme introduces a bias since it yields the property that for any path p and p', $l_p(d^{\min}) > l_{p'}(d^{\min}) \implies l_p(d^{\max}) > l_{p'}(d^{\max})$. Consequently, we also use a more general interval generation scheme for an activity j, using two parameters α and β, such that

$$\left[d_j^{\min} \in \{10, \alpha 50\}, d_j^{\max} \in \left\{d_j^{\min}, (1+\beta) d_j^{\min}\right\}\right].$$

Parameters α and β have been picked in the set $\{0.3, 0.6, 0.9\}$, the possible combinations defining, for each order strength, 9 set of 100-activities instances denoted $\alpha - \beta-$instances.

5.2 Algorithm comparison

We first present a comparison between the two path algorithms, the original and the improved one on the $PSPLIB-$ and $RG-$ instances. The computing times (in seconds) compiled in Table 1 confirm the theoretical dominance of our variant of the Path algorithm. Computing times are divided by almost 2 on every problem instance. Furthermore for the instnaces with $OS = 0.9$, the improved variant is able to obtain solutions in less than 2 hours while the original variant cannot. For the remaining of the tests, we keep only the improved variant of the path algorithm.

The two MIP formulations are compared in Table 2 on the $PSPLIB-$ and $RG-$instances. The activity-dependent MIP runs faster on all tested instances. On the most difficult instances with order strength at 0.8, MIPd needs 98.80 seconds to compute all the floats whilst MIPi needs 197.49 in average and 671.45 in the worst case. The high number of iterations of MIPi – at least 50 – is the main reason of the MIPd dominance, since each iteration of MIPi is expected to be harder to solve. In the following, MIPi results are discarded and only MIPd is compared to other algorithms.

Table 3, reports for each algorithm (improved_path, MIPd, BB), the average and maximal computing times grouped by series of instances. CPU times were limited to one hour. The $\alpha - \beta$-instances are grouped by order strength and β values since α is not discriminant for all algorithms.

The MIPd algorithm is dominated by the Path algorithm on all series of instances but the 0.9 OS series. Actually, the Path algorithm fails to compute minimal floats after one hour. The results obtained on the

serie	average time sec.		maximum time	
	original	improved	original	improved
$PSPLIB$				
$OS \in [0.1, 0.5]$	0.01	< 0.005	0.03	0.01
RG				
$OS = 0.1$	0.00	0.00	0.01	0.01
$OS = 0.2$	0.01	0.00	0.02	0.01
$OS = 0.3$	0.03	0.02	0.04	0.02
$OS = 0.4$	0.07	0.04	0.11	0.05
$OS = 0.5$	0.21	0.12	0.32	0.19
$OS = 0.6$	0.72	0.43	1.02	0.62
$OS = 0.7$	4.02	2.50	6.03	3.79
$OS = 0.8$	52.71	33.96	90.98	55.99
$OS = 0.9$	5465.00	2843.00	10524.00	6726.00

Table 1: Comparison of the two path algorithms

serie	average time sec.		maximum time		average iterations	
	MIPi	MIPd	MIPi	MIPd	MIPi	MIPd
$PSPLIB$						
$OS \in [0.1, 0.5]$	22.34	4.55	130.84	23.02	50	120
RG						
$OS = 0.1$	4.65	0.25	8.89	0.33	83	100
$OS = 0.2$	12.28	0.54	17.93	0.69	81	100
$OS = 0.3$	19.37	1.08	26.85	1.61	80	100
$OS = 0.4$	28.21	2.29	43.52	3.40	77	100
$OS = 0.5$	44.62	4.99	110.74	7.71	75	100
$OS = 0.6$	70.42	9.22	169.90	16.51	73	100
$OS = 0.7$	140.19	14.95	255.91	21.13	70	100
$OS = 0.8$	197.49	23.12	671.45	98.80	66	100
$OS = 0.9$	99.52	17.43	430.94	111.45	58	100

Table 2: Comparison of MIP formulations

$PSPLIB-$instances reveal that the MIPd suffers a lot of the increase of the number of activities. The BB procedure outperforms the other algorithms. The dominance of the BB algorithm increases with the order strength.

We can also state that $RG-$instances can be quickly solved by the MIPd algorithm, and to a lesser extent by the BB algorithm. Indeed, the MIPd algorithm effort increases with the order strength and β values. It requires at the worst case 78.14, 590.17 and 1776.73 seconds to solve $\alpha - \beta-$instances with $\beta = 0.3$, 0.6 and 0.9, respectively. MIPd has similar results on $PSPLIB-$, $RG-$ and $\alpha - \beta = 0.3-$instances where the ratio between d^{max} and d^{min} is close to 1.2. In those sets of instances, the dominance between paths can easily overpass scenarios because of their similar duration increase from d^{min} to d^{max}. We assume that this property can strengthen the branch-and-bound procedure in the MIP solver.

The $\alpha-\beta-$instances are derived from the $RG-$instances and their transformation keeps the underlaying network unchanged. The Path algorithm therefore has the same performance on every duration interval patterns of each instance. Regarding the number of nodes explored, we also found that the results are similar for the BB algorithm on every duration interval patterns of each instance. For those reasons the results concerning different series of instances are merged in the Table 4, which gives the average and maximum number of paths and the number of nodes explored by the Path and BB algorithms, respectively.

As expected, the Path algorithm is able to solve quickly instances with low-to-medium order strength (up to 0.7). Considering the order strength definition and the Table 4, it is obvious that a high order strength enforces a high number of paths which is the critical aspect of the Path algorithm. We remark that the maximum number of nodes explored in BB is not so sensitive to the order strength, smoothly varying from 2279 to 87113. It means that the dominance rules implemented in BB are strong enough to prune many scenarios. Computing times of BB slightly increase with the order strength – it run from 0.01 to 0.08 seconds – since, in the worst case, all the paths of the network are visited. However, the number of explored nodes in average is surprisingly decreasing from $OS = 0.8$ to $OS = 0.9$. We explain that phenomenon by the following reasoning. Let us consider an optimal scenario for the minimum float of the activity i. Let p be the longest path traversing i and p' the longest path for the extreme scenario p. The paths p' and p diverge at the activities a and b. As the activity-network order strength is increasing, the probability to find a and b close to i is also increasing. As the BB algorithm starts to evaluate pairs of possible diverging activities from i to the extremities 0 and $N + 1$, we can expect to find optimal floats – then cut many nodes – early.

Though the BB procedure seems to be sensitive to the order strength, we can not state on the impact of the values of parameters α and β. In our implementation of BB, no rule guides the exploration of activities (the list

serie	average time sec.			maximum time		
	Path	MIPd	BB	Path	MIPd	BB
PSPLIB						
$OS \in [0.1, 0.5]$	< 0.005	4.55	< 0.005	0.01	23.02	0.01
RG						
$OS = 0.1$	< 0.005	0.25	< 0.005	0.01	0.33	0.01
$OS = 0.2$	< 0.005	0.54	< 0.005	0.01	0.69	0.01
$OS = 0.3$	0.02	1.08	< 0.005	0.02	1.61	0.01
$OS = 0.4$	0.04	2.29	< 0.005	0.05	3.40	0.01
$OS = 0.5$	0.12	4.99	0.01	0.19	7.71	0.02
$OS = 0.6$	0.43	9.22	0.01	0.62	16.51	0.02
$OS = 0.7$	2.50	14.95	0.02	3.79	21.13	0.04
$OS = 0.8$	33.96	23.12	0.02	55.99	98.80	0.04
$OS = 0.9$	> 1800	17.43	0.02	> 1800	111.45	0.03
$\alpha - \beta = 0.3$						
$OS = 0.1$	< 0.005	0.24	< 0.005	0.01	0.33	0.01
$OS = 0.2$	< 0.005	0.51	< 0.005	0.01	0.67	0.01
$OS = 0.3$	0.02	1.03	< 0.005	0.02	1.58	0.01
$OS = 0.4$	0.04	2.17	< 0.005	0.06	3.6	0.01
$OS = 0.5$	0.12	4.86	0.01	0.22	8.64	0.02
$OS = 0.6$	0.44	9.04	0.01	0.62	14.07	0.02
$OS = 0.7$	2.51	15.65	0.02	3.83	24.78	0.04
$OS = 0.8$	34.37	24.62	0.02	58.01	73.29	0.06
$OS = 0.9$	> 1800	17.12	0.02	> 1800	78.14	0.04
$\alpha - \beta = 0.6$						
$OS = 0.1$	< 0.005	0.26	< 0.005	0.01	0.35	< 0.005
$OS = 0.2$	< 0.005	0.56	< 0.005	0.01	0.72	0.01
$OS = 0.3$	0.02	1.15	< 0.005	0.02	1.71	0.01
$OS = 0.4$	0.04	2.37	< 0.005	0.05	3.67	0.01
$OS = 0.5$	0.12	5.34	0.01	0.17	8.08	0.02
$OS = 0.6$	0.43	10.71	0.01	0.65	17.49	0.02
$OS = 0.7$	2.5	23.43	0.02	4.03	52.17	0.04
$OS = 0.8$	34.35	63.89	0.02	56.45	157.07	0.06
$OS = 0.9$	> 1800	94.61	0.02	> 1800	590.17	0.04
$\alpha - \beta = 0.9$						
$OS = 0.1$	< 0.005	0.27	< 0.005	< 0.005	0.34	0.01
$OS = 0.2$	< 0.005	0.6	< 0.005	0.01	0.82	0.01
$OS = 0.3$	0.02	1.25	< 0.005	0.02	1.85	0.01
$OS = 0.4$	0.04	2.55	< 0.005	0.05	3.82	0.01
$OS = 0.5$	0.12	5.77	0.01	0.22	9.49	0.02
$OS = 0.6$	0.43	12.12	0.01	0.66	23.89	0.02
$OS = 0.7$	2.52	34.76	0.01	3.87	84.68	0.03
$OS = 0.8$	34.38	124.22	0.02	57.71	267.88	0.05
$OS = 0.9$	> 1800	338.98	0.01	> 1800	1776.73	0.08

Table 3: Computing times comparison on all instances

$\mathcal{P}$ is unsorted). We can imagine that such rules can speed-up the procedure exploiting the regularity of the instances with low value of α and β.

6 Conclusion

We have proposed a very fast method for computing the minimum float of activities in activity-on-node networks, under interval duration uncertainty. Thanks to newly established structural properties of optimal solutions and lower-bound on partial solutions, the method consistently outperforms the methods it has been compared with: new mixed integer programming formulations solved by a commercial solver and the path enumeration algorithm of Dubois *et al.* [5] and Fortin *et al.* [7].

References

[1] F. Ballestín. When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166, 2007.

[2] S. Chanas and P. Zielínski. The computational complexity of the criticality problems in a network with interval activity times. *European Journal of Operational Research*, 136:541–550, 2002.

[3] S. Chanas and P. Zielínski. On the hardness of evaluating criticality of activities in planar network with duration intervals. *Operation Research Letters*, 31:53–59, 2003.

[4] E. Demeulemeester, M. Vanhoucke, and W. Herroelen. Rangen: a random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1):17–38, 2003.

[5] D. Dubois, H. Fargier, and J. Fortin. Computational methods for determining the latest starting times and floats of tasks in interval-valued activity networks. *Journal of Intelligent Manufacturing*, 16:407–422, 2005.

[6] D. Dubois, H. Fargier, and V. Galvagnon. On latest starting times and floats in activity networks with ill-known durations. *European Journal of Operational Research*, 147:266–280, 2003.

[7] J. Fortin, P. Zieliński, D. Dubois, and H. Fargier. Criticality analysis of activity networks under interval uncertainty. *Journal of Scheduling*, 2010. DOI: 10.1007/s10951-010-0163-3.

[8] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2):289–306, 2005.

[9] Adam Kasperski. *Discrete Optimization with Interval Data*. Springer, 2008.

[10] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Springer, 1997.

[11] R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems i – general strategies. *Mathematical Methods of Operations Research*, 28(7):193–260, 1984.

serie order strength	Paths explored by Path avg.	 max.	Nodes explored by BB avg.	 max.
$PSPLIB$				
$OS \in [0.1, 0.5]$	326	1277	5352	12422
RG and $\alpha - \beta$				
$OS = 0.1$	404	438	1806	2279
$OS = 0.2$	804	890	3253	4245
$OS = 0.3$	1644	1929	5160	7066
$OS = 0.4$	3793	4783	8161	12253
$OS = 0.5$	10527	14208	12265	21144
$OS = 0.6$	38024	50672	16970	30696
$OS = 0.7$	228018	346082	21191	47532
$OS = 0.8$	3325121	5595599	23131	75232
$OS = 0.9$	298930149	707330531	18063	87113

Table 4: Number of paths and nodes explored by Path and BB algorithms, respectively